SDEV 1201 Database Fundamentals

LESSON 6

Query Syntax

Queries are written with the SELECT statement. The simplified syntax for a select statement is:

```
SELECT [ALL | DISTINCT] column_list
[FROM table_name [, table_name2 […, table_name]] -- source of data
          [INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN] -- other sources
[WHERE clause] -- filter
[GROUP BY clause] -- aggregate by
[HAVING clause] -- filter by aggregate
[ORDER BY clause] -- sorts results
```

If it looks complex don’t worry. All the “clauses” listed don’t need to be used when creating a query. However, what clauses you do use MUST be in the order shown above (i.e. HAVING MUST be after GROUP BY for example and not vice versa).

Simple SELECT

```
SELECT
    FirstName
, LastName
, City

FROM Student;
```

Like

The Like operator is used when you use wildcards in your search or perform pattern matching. Wildcards are characters that when used with the Like operator allow the wildcards to represent a character or multiple characters. The Like operator is case sensitive in PostgreSQL.

For example, if we wanted to return all the student's names whose names started with the letter 'D' we would use the % wildcard character. The % wildcard means any character and any number of characters.

```
Select FirstName || ' ' || LastName "Student Name"
From Student
Where FirstName Like 'D%';
```

It is important to use the like operator when using wildcards. The % would be interpreted as a literal '%' if we used = instead of Like.

Like

The _ (underscore) is a single character wildcard. This would be used when you are looking for any one character in your search criteria. If we wanted the students' names whose first name was 3 characters long and started with 'Ja' we could use this query.

```
Select FirstName || ' ' || LastName "Student Name"
From Student
Where FirstName Like 'Ja_';
```

If we had used a % instead of _ we would also return students whose First Name was longer than 3 characters.

Aggregate Functions

Aggregate functions calculate a single value using the data from many records. If you have used function in Excel such as SUM, AVERAGE, COUNT, etc…. then you have used aggregate functions.

```
aggregate_function_name ([ ALL | DISTINCT ] expression)
```

- `AVG(ColumnName)` returns the average of numeric values. `NULL` is ignored.
- `SUM(ColumnName)` returns the sum of a column containing numeric values.
- `MIN(ColumnName)` and `MAX(ColumnName)` returns the minimum & maximum values from a column of numeric, date, or character values.
- `COUNT(*)` returns the number of records that match the `WHERE` criteria.
- `COUNT(ColumnName)` returns the number of non-null values in a given column for records that match the `WHERE` criteria.

String Functions

- LENGTH(*column* | *expression*)
- LEFT(*column* | *expression*, *length*)
- RIGHT(*column* | *expression*, *length*)
- SUBSTRING(*column* | *expression*, *start*, *length*)
- REVERSE(*column* | *expression*)
- UPPER(*column* | *expression*)
- LOWER(*column* | *expression*)
- LTRIM(*column* | *expression*)
- RTRIM(*column* | *expression*)

String Functions

- LENGTH(*column* | *expression*)
- LEFT(*column* | *expression*, *length*)
- RIGHT(*column* | *expression*, *length*)
- SUBSTRING(*column* | *expression*, *start*, *length*)
- REVERSE(*column* | *expression*)
- UPPER(*column* | *expression*)
- LOWER(*column* | *expression*)
- LTRIM(*column* | *expression*)
- RTRIM(*column* | *expression*)

Date Functions

- `Current_Date` returns the current system date
- `Current_Time` returns the current system time
- `DATE_PART(xx, date1)` returns integer representation of the `xx` of `date1`

`xx` represents field for date portions (see next slide).

Date Function Examples

-- You can also use Fields to get the character representation of dates.

```
Select To_Char (Current_Date, 'DAY');
```

Returns the day of the week (i.e. MONDAY, TUESDAY, etc.). Note that the "Field" is case sensitive: "Day" would return "Monday".

```
Select To_Char (Current_Date, 'MONTH');
```

Returns the month name. Note that the "Field" is case sensitive: "Month" would return "September".

Group By

The GROUP BY clause is used with aggregate functions to provide subtotals. We could easily return the average Mark from the grade table. What if we wanted the average Mark for each course? For each student? For each course for each Student? To do this we would use GROUP BY.

```
SELECT StudentID, AVG(Mark) AS AverageMark
FROM Registration
GROUP BY StudentID
```

calculates the average mark per Student.

We will usually GROUP BY something unique.

Another way to think of the syntax GROUP BY is the 'for each'.

Having

HAVING is like the WHERE clause, except it applies its criteria after GROUP BY. For example:

```
SELECT StudentID, AVG(Mark) as AverageMark
FROM Registration
GROUP BY StudentID
WHERE AVG(Mark) > 80
```

Doesn't work. However, Replacing WHERE with HAVING having does.

```
SELECT StudentID, AVG(Mark) as AverageMark
FROM Registration
GROUP BY StudentID
HAVING AVG(Mark) > 80
```

Today’s Objective

By the end of this section, you will be able to:

- Table joins
- Table Joins with Aggregates

Theory and reference material for this presentation is based on the “Advanced Select Statement” document in the “Week 4 – Select Statements” section of Brightspace.

Joins

Joins

To work with data that is located in more than one table you need to instruct the server how to relate records from one table with its related record(s) in other tables. To do this you must use the join operator. The join operator lets us control how the server relates rows from different tables.

If you wanted to select all the staff names and their position descriptions you would need to select the staff names from the staff table and the position descriptions from the position table.

According to the syntax of a Select statement you could write a select statement as follows:

```
select FirstName || ' ' || LastName "Staff Name", PositionDescription "Position "
from Staff, Position
```

Joins

When you run this Select statement you return 70 records. This may seem like the correct select statement as there were no errors. However, closer inspection of the data in the two tables involved leads to some questions.

First, the Staff table contains only 10 records. So, we should only be seeing 10 records coming back (select all the staff names and their position descriptions).

Second, the position table has 7 records (there are 7 positions).

So, where did the 70 records come from? If you look at the results of the select closely you will notice that every staff member seems to have every position. 10 Staff X 7 Positions = 70.

Joins

What happened was there was nothing in the select statement telling the server how to relate what records on the Position table relate to what records in the Staff table.

Remember, records in tables relate to each other through the foreign key. The value of the primary key in the parent table relates to records in the child table that have the same value. Staff that are Position 2 (Program Chair) will have a value of 2 in the PositionID field which is the foreign key in the Staff Table.

Obviously, every staff member is not in every position at the school so we need to instruct the server which field in the parent table relates to what field in the child table. This is usually the Primary Key in one table and the Foreign Key in another. This is the Parent field in the parent table and the foreign key in the child table. In this case, it is the PositionID which joins the Position and Staff table and forms the relationship. It also identifies what records in the parent relate to what records in the child.

Joins

To ensure we get only the Parent records associated with their correct child records we must use syntax which identifies what the common (join) field is between the tables.

Syntax:

SELECT field1, field2,… FROM table1

INNER JOIN table2

ON table1.joinfield = table2.joinfield

Joins

For example, if I want to select some fields from the Staff table and other fields from the Position table, which are connected using the PositionID PK/FK

```
select FirstName || ' ' || LastName "Staff Name",
PositionDescription "Position"
from Staff
inner join Position
on Staff.PositionID = Position.PositionID
```

The INNER JOIN syntax tells the server that we are performing an inner join and identifies that PositionID is the field in the 2 tables that relate them together.

Joins

An Inner Join only returns records from the parent table that have related records in the child table. In this example Position is the Parent table (Primary Key in the relationship) and Staff is the child table (has the foreign key in the relationship). Looking back at the results you will notice that one Position Description is not returned in the results. Position 7 (Assistant Dean) was not returned because there are no Staff with that position. With an Inner Join parent records that do not have a matching child record are not returned.

SDEV1201 Coding Standard

When doing an inner join the syntax “inner join” must be used for clarity.

Joins

If I want to select some fields from the Student table and other fields from the Registration table, which are connected using the StudentID PK/FK:

```
SELECT FirstName, LastName, CourseID, Mark
FROM Student
INNER JOIN Registration
ON Student.StudentID = Registration.StudentID
```

The INNER JOIN syntax tells the server that we are performing an inner join and identifies that StudentID is the field in the 2 tables that relate them together.

Joins

If we want to select the student names from the Student table and the payment dates and amounts from the Payment table, which are connected using the StudentID PK/FK:

```
SELECT FirstName, LastName, PaymentDate, Amount
FROM Student
INNER JOIN Payment
ON Student.StudentID = Payment.StudentID
```

We join on StudentID because it is the primary key of the Student table and a foreign key in the Payment table.

Joins

Consider the following Select statement which is now selecting the StudentID field as well:

```
select StudentId, FirstName || ' ' || LastName "Student Name", PaymentDate
from Student
inner join Payment
on Student.StudentID = Payment.StudentID
```

This will result in an error:

```
ERROR: column reference "studentid" is ambiguous
```

This error is due to the fact that we must state the table that we are selecting the fields from. Since StudentID is in both the Student and the Payment table, the server does not know which one to select.

Joins

To solve this we need to adjust our syntax to identify which table the server should select the ambiguous column from.

```
select Student.StudentId, FirstName || ' ' || LastName "Student Name",
PaymentDate
from Student
inner join Payment
on Student.StudentID = Payment.StudentID
```

While it does not usually matter which table you select the ambiguous columns from it is almost always the parent table.

You can also encounter this error when you have an ambiguous column in other parts of a select statement (like where).

Selecting from 3 or more tables

To select from more than 2 tables the syntax would look like:

```
SELECT field1, field2,... FROM table1
INNER JOIN table2
ON table1.joinfield = table2.joinfield
INNER JOIN table3
ON table.joinfield = table.joinfield
INNER JOIN table4
ON table.joinfield = table.joinfield
```

Selecting from 3 or more tables

Select the Student Names, Payment Dates, and Payment Type Descriptions for all the students that have payments:

```
select Student.StudentId, FirstName || ' ' || LastName "Student Name",
PaymentDate, PaymentTypeDescription
from Student
inner join Payment
on Student.StudentID = Payment.StudentID
inner join PaymentType
on Payment.PaymentTypeID = PaymentType.PaymentTypeID
```

Break

Practice

- Work on the “Inner Join Exercise” exercise found under “Week 4” in the Brightspace site.

- For reference please see the “Advanced Select Statement” document in the “Week 4 – Select Statements” section of Brightspace.

Joins With Aggregates

Now that we understand joins, we have much more avenues to explore the data and retrieve relevant information. With our IQSchool database, what if we would like to know the number of times each course is offered.

```
select CourseName, count(Registration.CourseID) "Number of times the course is offered"
from Course
inner join Registration
on Course.CourseId = Registration.CourseId
Group By Course.CourseId, Course.CourseName
```

Joins With Aggregates

Another request has been made for the student names and total sum of payments each student has made.

```
select FirstName ||  ' ' || LastName "Student Name",
sum(Amount) "Total amount paid“
from Student s
inner join Payment p
on s.StudentID = p.StudentID
group by s.StudentID, s.FirstName, s.LastName
```

Practice

- Please work on the Inner Join With Aggregates Exercises found in the “Week 4” section on the Brightspace site.
- Complete Inner Join Exercise found in the “Week 4” section on the Brightspace site.

NOTE: **“Basic Select Statement” assignment** is **due** on **Friday, September 26, 2025 at 5:00 PM**.

Coding Standards (as found in the course syllabus)

Only those database programming methods, practices, and techniques taught in class, or available from Brightspace notes and/or exercises, will be permissible in SDEV1201 labs and quizzes. Any solution in a lab or quiz submission using non-approved methods, practices, and/or techniques, or code that is not executable in a Postgres environment, will be given a mark of zero.